

ToDo-List

Personal Notes Management Mobile Application (Cross-Platform)

- Technical Report -

Project Type: Build

Project Date: 31st December 2021

Table of Contents

Table of Contents	1
List of Figures/Tables	3
1.0 Mobile Application Development.....	5
1.1 Mobile Application Categories	5
1.2 Demanded Development Type Justification	8
2.0 ToDo-List Application.....	9
2.1 Introduction.....	9
2.2 Features	9
2.3 Architecture.....	10
2.4 UI/UX Design	12
2.5 Implementation	20
2.6 Test Case.....	33
References.....	38

List of Figures

Figure 1: Mobile Application Development	5
Figure 2: Wireframe - View list of existing created notes	12
Figure 3: Wireframe - Create new note	13
Figure 4: Wireframe - Edit existing note	14
Figure 5: Wireframe - Search among existing notes.....	15
Figure 6: Implemented Design - View list of existing created notes	16
Figure 7: Implemented Design - Create new note	17
Figure 8: Implemented Design - Edit existing note	18
Figure 9: Implemented Design - Search among existing notes.....	19
Figure 10: Interface: View existing notes	20
Figure 11: Interface: Create new note.....	21
Figure 12: Interface: Edit existing note.....	22
Figure 13: Interface: Search existing notes.....	23
Figure 14: Folder structure.....	24
Figure 15: Code - main.dart	24
Figure 16: Code - notes_log.dart #1.....	25
Figure 17: Code - notes_log.dart #2.....	26
Figure 18: Code - update_note.dart #1.....	27
Figure 19: Code - update_note.dart #2.....	28
Figure 20: Code - search.dart.....	29
Figure 21: Code - note_design.dart.....	30
Figure 22: Code - database.dart	31
Figure 23: Code - load.dart	32
Figure 24: View list of existing notes - Test Evidence	35
Figure 25: Create new note - Test Evidence #2	36
Figure 26: Create new note - Test Evidence #1	36
Figure 27: Edit existing note - Test Evidence #2.....	37
Figure 28: Edit existing note - Test Evidence #1	37
Figure 29: Delete existing note - Evidence #2.....	38
Figure 30: Delete existing note - Evidence #1	38

List of Tables

Table 1: Selected tools and technologies	10
Table 2: Test case template	33
Table 3: Testing targets.....	34
Table 4: Test Case 01 - View list of existing notes.....	35
Table 5: Test Case 02 - Create new note	36
Table 6: Test Case 03 - Edit existing note	37
Table 7: Test Case 04 - Delete existing note	38

1.0 Mobile Application Development

Mobile applications are softwares which are specifically designed to run on small-sized wireless devices like smartphones and tablet computers. These devices are well known for their simpler accessibility and portability. Mobile applications generally provide less features compared to web and desktop applications. These applications are developed focusing on the demanded features of any software, and are based to be accessed easier compared with web or desktop functionalities. These applications are designed to access the device's hardware components such as microphone, speaker and sensors to provide additional features, as mobile devices contain more hardware mechanisms by default compared to computers [1].



Figure 1: Mobile Application Development

1.1 Mobile Application Categories

Mobile applications are mainly categorized into four types,

1. Native
2. Cross
3. Web-based
4. Hybrid

01. Native

Native mobile application development means that the software would be developed to run based on one specific mobile operating system [2]. The two major mobile operating systems in the market are Google's Android, Apple's iOS and Microsoft's Windows. These different devices have different methodologies and approaches of development. As native development focuses on a specific platform, the application would have better performance and provides a better user experience for usage.

Native applications would also be more responsive to user interactions, as they are compiled in consideration of one platform. Moreover, native applications would also be more secure, as different mobile operating systems have different security mechanisms meaning the implement approach based on one specific platform would be safer in general.

02. Cross

Cross-platform mobile applications are developed to be compatible on multiple operating system platforms, to increase effectiveness [3]. These applications are developed in a manner to be executable on various operating systems mainly including iOS, Android and Microsoft Windows. The concept is that developers would find it easier and efficient to implement a single application to be able to run on multiple operating system platforms. This would make the development procedure hassle-free.

The main advantage of cross-platform development is that the source code can be shared between multiple platforms, which reduces the workload on developers and also allows to control the application budget. Application owners would be able to advertise to a wide range of mobile users as there is no need to focus on different development platforms. Users from a wide range of operating systems would be able to access the application without any platform-based constraints or limitations.

03. Web-based

Web-based mobile applications are simply web applications that requires a web browser installed on the device. They would have the same functionalities when accessing through multiple platforms, but the design and orientation of the application could change based on the device display sizes [4]. Web-based development would be efficient for applications which have a web application as the central system. This type allows to save time and budget during the implementation procedure.

Web-based applications help developers to get rid of most development stress and time. If an application is supposed to be web-based, the developer's only requirement would be to develop a responsive web application. This would also be an advantage if the mobile application requirement is just supposed to be a part of the central web application. The contents of the mobile application would be the exact same of the web application, with just the orientation adjusted with the site responsiveness.

04. Hybrid

Hybrid mobile applications are basically web-based mobile applications developed in a native application. It needs to be locally installed in the device to run, but the elements in the application will be processed from a web application. This allows these applications to run properly on multiple mobile platforms [5]. Hybrid applications work using the shell of a native application. The contents of the required application will be displayed on the shell, which would be extracted from a web application.

Especially, the application user would not be able to identify if the application is based on complete native or web-driven. This development type is mostly suitable if the specific application doesn't need any key elemental differences between the web application and the mobile application. Users would be able to access the web contents in a native application itself. Hybrid applications reduce development costs and are also time-saving. Development resources are also relatively easier to identify.

1.2 Demanded Development Type Justification

Cross-platform mobile application development is highly demanded nowadays. Cross-platform mobile applications are designed to be compatible on multiple operating system platforms, to increase effectiveness. Along with functionalities, it is also important that applications can run efficiently on different device platforms to provide seamless experience to users [3]. Cross-platform development saves time for the software publishers as well as the developers. The developers are able to program the application once and then allow it to be installed and work equivalently on multiple platforms.

Moreover, cross-platform application development is relatively cheaper compared to native development. The application publisher would be required to pay for each native application dedicated to one platform, which requires multiple developments, making it more expensive and time-consuming. Even if the publisher recruit one developer, it would still be sufficient to implement and manage the system. End users would also not feel much difference between native and cross-platform applications.

Application users from multiple operating system platforms would be able to use the application, without considering any platform-based constraints or restrictions. If the application were to be developed on native-type, the developers would be required to consider various platform-based rules and regulations based on each operating system, which consumes more time as well as more time would be wasted in this procedure.

Considering businesses and start-up companies wanting to develop mobile applications, they would be able to attract a wider range of customer base with a relatively low budget by using cross-platform application development. This way, they would have the freedom to provide services for user on multiple device platforms using one application source code. Moreover, the application maintenance costs would also be cheaper compared to maintaining native applications for different platforms [6].

2.0 ToDo-List Application

2.1 Introduction

"ToDo-List" is a personal notes management application. Using this application, user can add new notes, update the notes, delete the notes, read the created notes, and search saved notes. The requirements for the implementation of the applications are,

- Use of suitable application architecture.
- Use of mobile development best practices for development.
- Use of UI (User Interface) / UX (User Experience).
- Use of suitable embedded database.

2.2 Features

01. Create new note.

This feature would allow the application user to create a new note in the “ToDo-List” application, based on the choice of the user. This feature would require the user to enter the desired ‘title’ of the note and then the ‘content’ related with the specific note to create a new note.

02. View existing notes.

This feature would act as the primary requirement of the application. This feature allows the user to view the list of all created notes in a list view, which would be the home screen of the application.

03. Edit existing note.

This feature would assist the user to update changes in any existing note. The user would be able to edit the existing note ‘title’ as well as its ‘content’ and then save the changes of the specific note.

04. Delete existing note.

This feature allows the user to delete any unnecessary notes created in the application. The user would be able to select any existing note from the ‘view existing notes’ section and delete the specific note.

2.3 Architecture

The selected system architectures for the development of ToDo-List application are listed below.

Table 1: Selected tools and technologies

Application State	Cross-Platform (iOS and Android)
Front-End Language	Dart
Front- End Framework	Flutter
Database System	SQLite Relational Data Management System
Development Environment	Android Studio and Visual Studio Code

01. Language: Dart

Dart programming language has been selected to program the requirement ToDo-List application. Dart is a free and open-source OOP (Object-Oriented Programming) language, consisted with C-based composition developed by Google LLC. Dart is majorly used to design the frontend user interfaces for mobile applications. Moreover, Dart is also a compiled language, the code requires to be compiled to machine code before being executed during development. [7]

02. Framework: Flutter

Flutter framework has been selected to assist the application development with Dart language. Flutter is an open-source and free mobile application user interface framework released by Google LLC company. Flutter is majorly popular among developers due to its ability to design a native mobile application using just one codebase. Eventually, this means that developers would find it simpler and more efficient to develop applications based on iOS and Android. Moreover, the Flutter framework is consisted of an SDK (Software Development Kit), which assists with tools used for development. It also consists a separate widget-based UI (User Interface) library, which contains a group of reusable UI elements. [7]

03. State Management: Cross-Platform

The ToDo-List application will be implemented using cross-platform development technique. Cross-platform is a development methodology procedure that allows application implementation to be able work on multiple mobile operating systems concurrently, mainly including iOS, Android and Microsoft Windows. These applications are programmed with native codes along with independent codes supporting multiple platforms, which increases the development efficiency and effectiveness. [9]

04. Database: SQLite

The ToDo-List application would be developed using SQLite database system for the backend procedures. SQLite is a software library that would provide a relational database management system for the Dart + Flutter application. SQLite is popular among developers due to its serverless, self-contained and transactional abilities. Moreover, the database also has way less configuration requirements compared to similar database systems. [10]

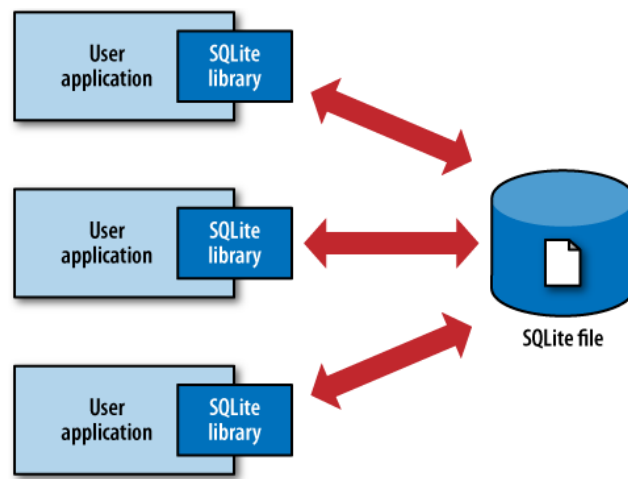


Figure 2: SQLite Structure

2.4 UI/UX Design

Wireframe User Interface Design:

01. View list of existing created notes (Home Page):

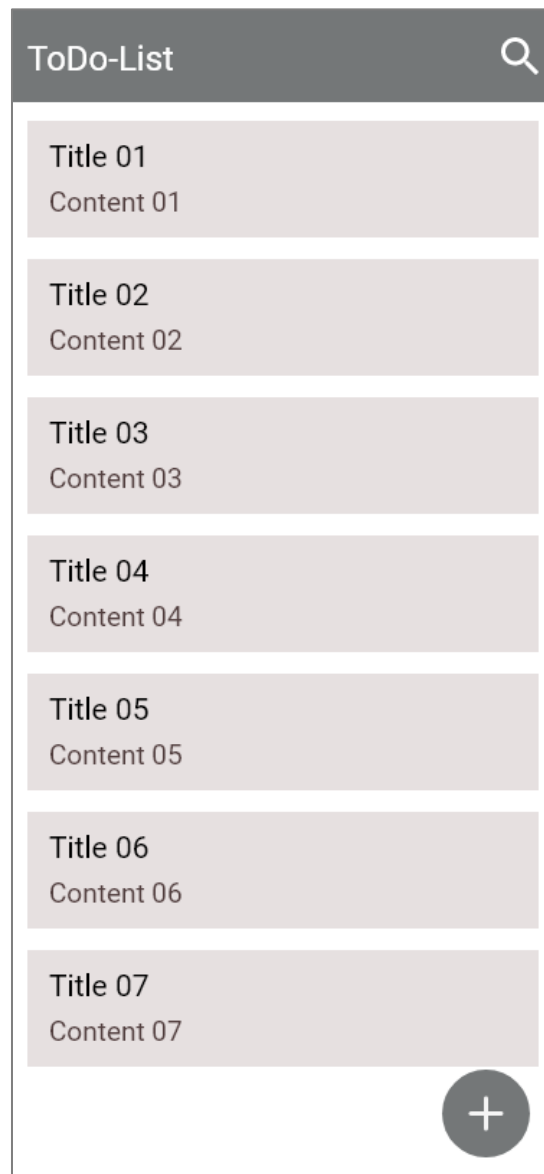
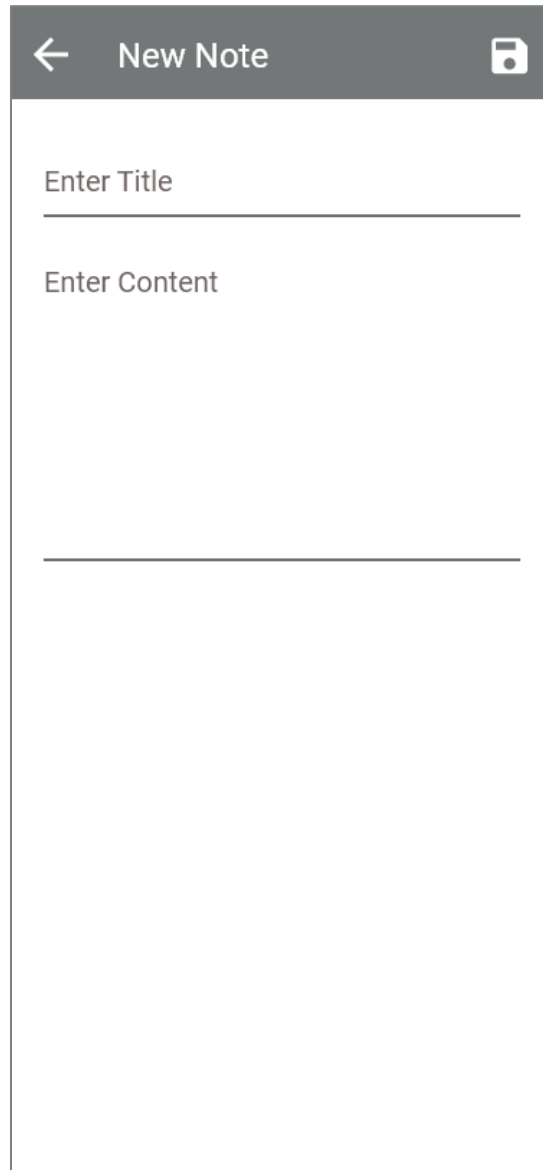


Figure 3: Wireframe - View list of existing created notes

02. Create new note:



A wireframe for a mobile application screen titled "New Note". The screen has a dark gray header bar with a white back arrow icon on the left, the text "New Note" in the center, and a white floppy disk icon on the right. Below the header, the main content area is white. It contains two input fields: the first is labeled "Enter Title" and has a horizontal line below it; the second is labeled "Enter Content" and has a horizontal line below it. The "Enter Content" field is significantly larger than the "Enter Title" field.

Figure 4: Wireframe - Create new note

03. Edit existing note:

← Edit Note

Existing Title

Existing Content

Figure 5: Wireframe - Edit existing note

04. Search among existing notes:

The wireframe shows a mobile application interface for searching through notes. At the top, there is a dark grey header bar. On the left side of this bar is a white back arrow icon. To the right of the arrow is a white rectangular search input field with the placeholder text "Search Notes" in a dark grey font. Below the header bar, the main content area is white and contains a vertical list of seven note items. Each item is enclosed in a thin black rectangular border and consists of two lines of text: a bold title followed by a regular weight content line. The titles are "Title 01" through "Title 07", and the content lines are "Content 01" through "Content 07" respectively. There is a small gap between each note item. At the bottom of the main content area, there is an additional empty rectangular box, suggesting a space for a new note or a confirmation area.

Back Arrow	Search Notes
Title 01 Content 01	
Title 02 Content 02	
Title 03 Content 03	
Title 04 Content 04	
Title 05 Content 05	
Title 06 Content 06	
Title 07 Content 07	

Figure 6: Wireframe - Search among existing notes

Implemented User Interface Design

01. View list of existing created notes (Home Page):

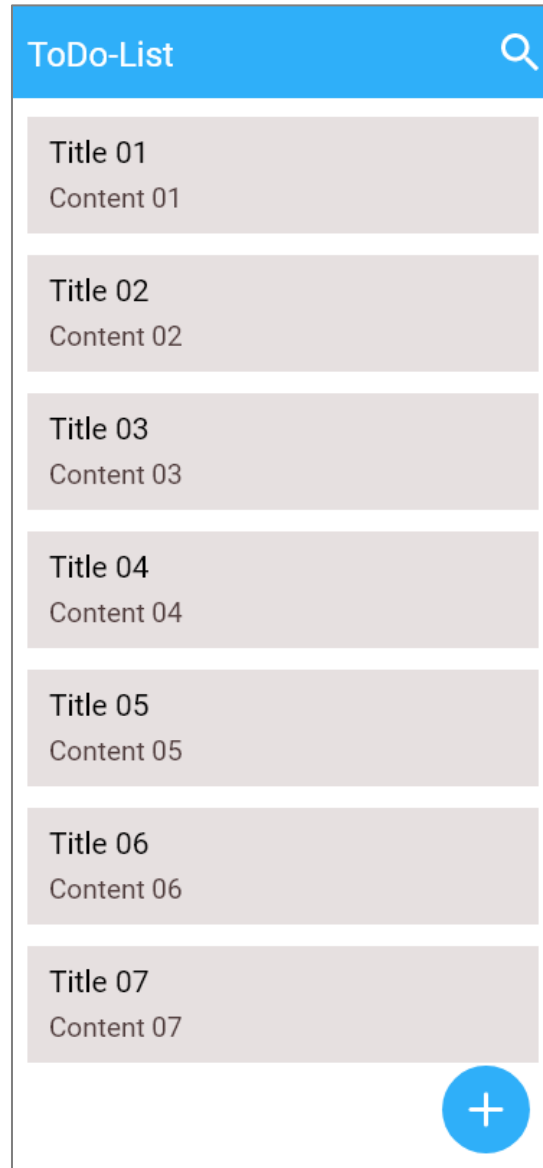
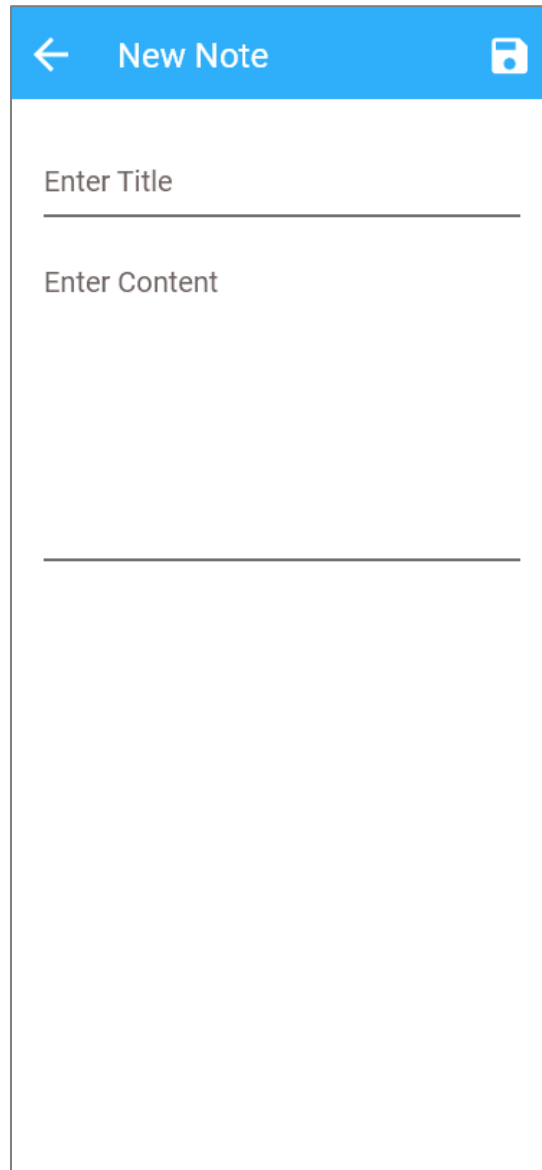


Figure 7: Implemented Design - View list of existing created notes

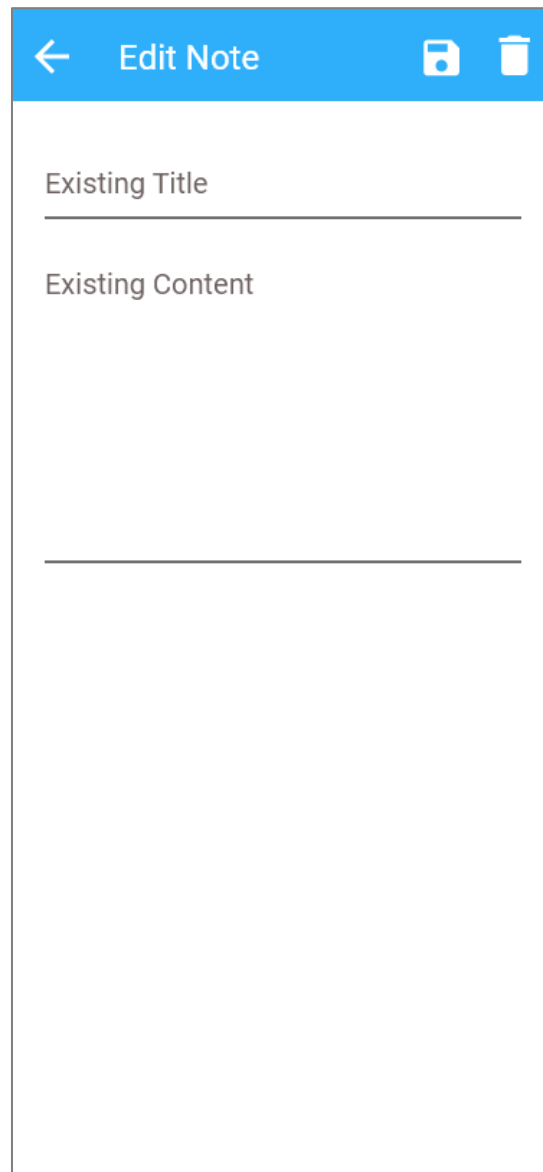
02. Create new note:



The image shows a mobile application interface for creating a new note. At the top, there is a blue header bar with a white back arrow on the left, the text "New Note" in the center, and a white floppy disk icon on the right. Below the header, the main area is white. It contains two input fields: the first is labeled "Enter Title" and has a horizontal line below it; the second is labeled "Enter Content" and has a horizontal line below it. The text "Enter Content" is positioned above a large, empty text area that occupies the bottom half of the screen.

Figure 8: Implemented Design - Create new note

03. Edit existing note:



The image shows a mobile application interface for editing a note. At the top, there is a blue header bar with a white back arrow on the left, the text "Edit Note" in the center, and two white icons on the right: a floppy disk (save) and a trash can (delete). Below the header, the main content area has a light gray background. It contains two text input fields. The first field is labeled "Existing Title" and has a horizontal line underneath it. The second field is labeled "Existing Content" and has a horizontal line underneath it. The rest of the content area is empty.

Figure 9: Implemented Design - Edit existing note

04. Search among existing notes:

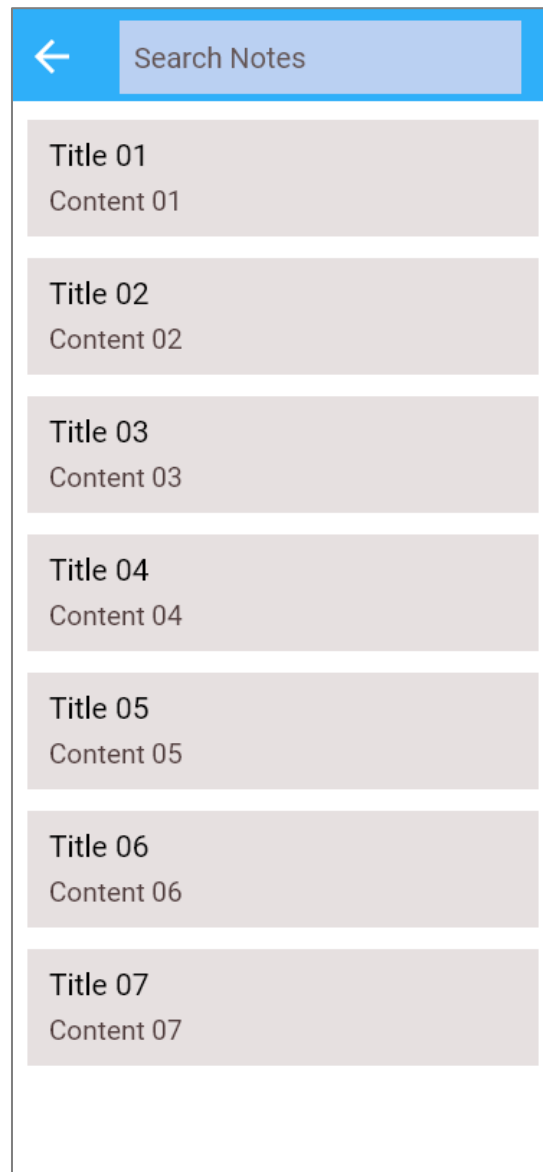


Figure 10: Implemented Design - Search among existing notes

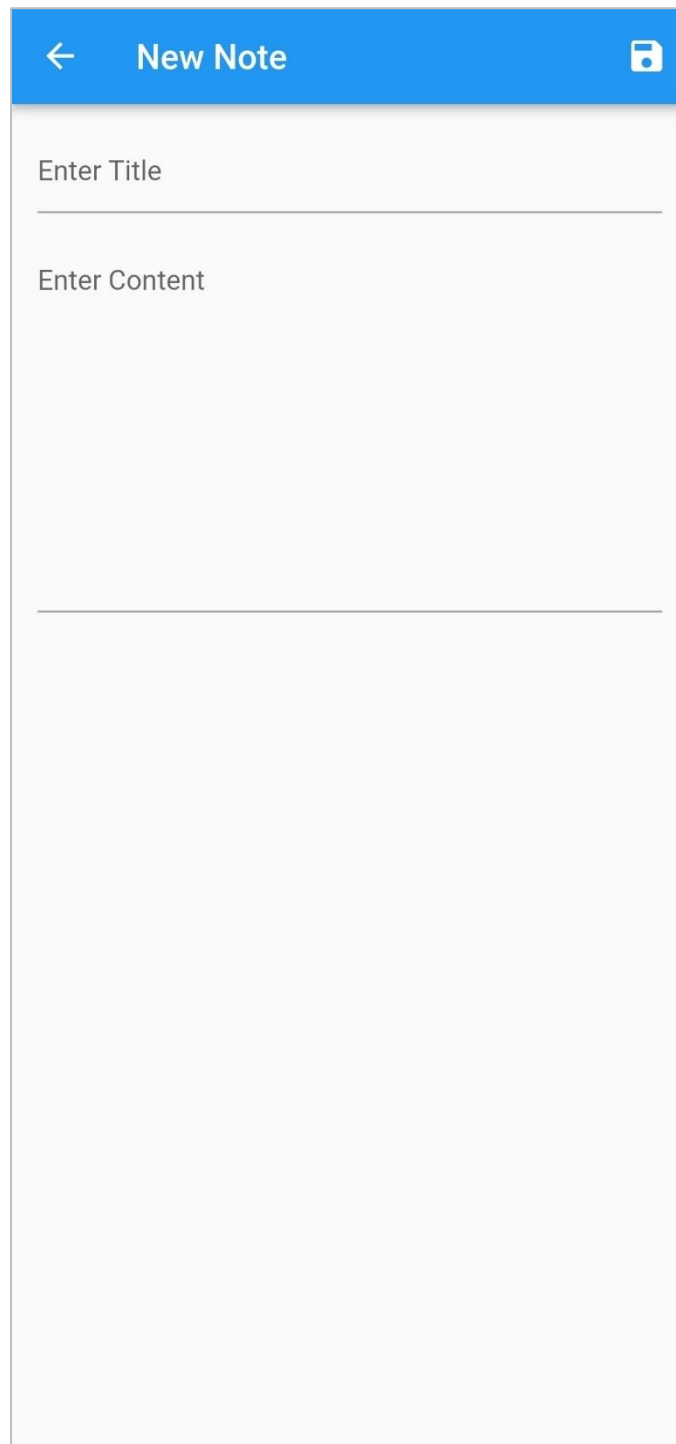
2.5 Implementation

View existing notes functional interface:



Figure 11: Interface: View existing notes

Create new note functional interface:



The image shows a mobile application interface for creating a new note. It features a blue header bar with a back arrow on the left, the text "New Note" in the center, and a save icon on the right. Below the header, there is a light gray area with two input fields. The first field is labeled "Enter Title" and has a horizontal line below it. The second field is labeled "Enter Content" and has a horizontal line below it. The rest of the screen is a large, empty light gray area for the note's content.

Figure 12: Interface: Create new note

Edit existing note functional interface:

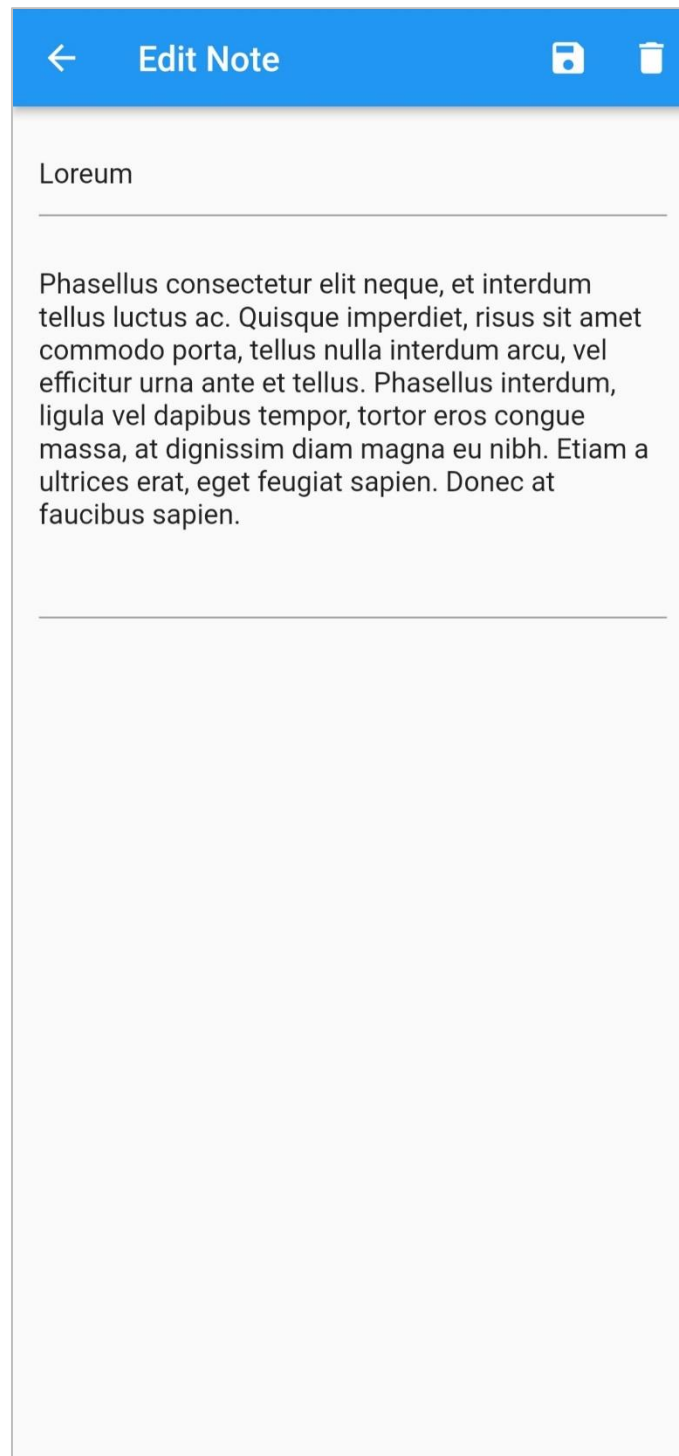


Figure 13: Interface: Edit existing note

Search existing notes application interface:

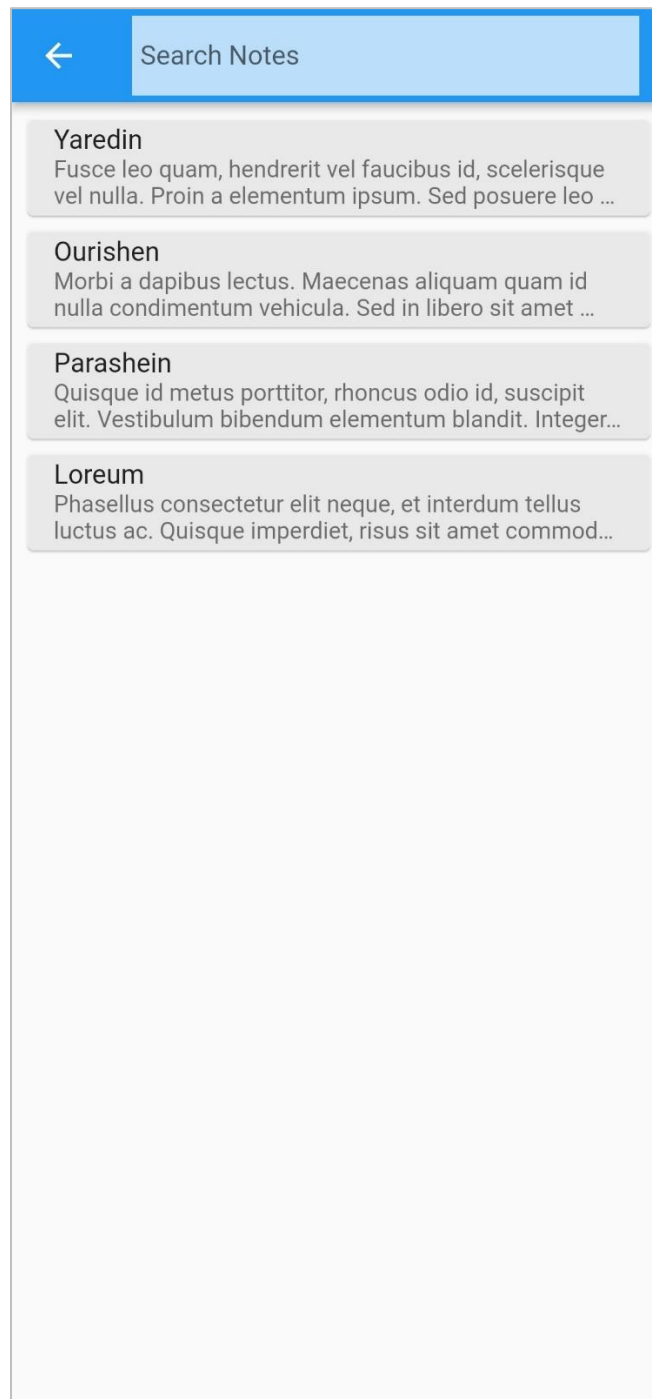


Figure 14: Interface: Search existing notes

Folder Structure:

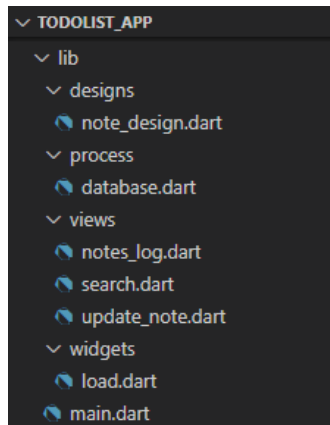


Figure 15: Folder structure

Codes snippets:

- lib > main.dart:

```
lib > main.dart > ...  
  
3  import 'package:flutter/material.dart';  
4  import 'package:todolist_app/views/notes_log.dart';  
5  
   Run | Debug | Profile  
6  void main() => runApp(ToDoList());  
7  
8  class ToDoList extends StatelessWidget {  
9  
10     // This widget is the root of your application.  
11     @override  
12     Widget build(BuildContext context) {  
13         return MaterialApp(  
14             title: 'ToDo-List',  
15             debugShowCheckedModeBanner: false,  
16             home: Home(),  
17         ); // MaterialApp  
18     }  
19 }
```

Figure 16: Code - main.dart

- lib > views > notes_log.dart:

```
lib > views > notes_log.dart > ...
3  import 'package:flutter/material.dart';
4  import 'package:todolist_app/designs/note_design.dart';
5  import 'package:todolist_app/process/database.dart';
6  import 'package:todolist_app/views/search.dart';
7  import 'package:todolist_app/views/update_note.dart';
8  import 'package:todolist_app/widgets/load.dart';
9
10 class Home extends StatefulWidget {
11   @override
12   HomeState createState() => HomeState();
13 }
14
15 class HomeState extends State<Home> {
16
17   List<Note> notes;
18   bool loading = true;
19
20   @override
21   void initState() {
22     super.initState();
23     refresh();
24   }
25
26   @override
27   Widget build(BuildContext context) {
28
29     return Scaffold(
30       appBar: AppBar(
31         title: Text('ToDo-List'),
32
33         actions: <Widget>[
34           IconButton(
35             icon: Icon(Icons.search),
36             onPressed: () {
37               Navigator.push(
38                 context,
39                 MaterialPageRoute(builder: (context) => Search()),
40               );
41             },
42           ), // IconButton
43         ], // <Widget>[]
44       ), // AppBar
```

Figure 17: Code - notes_log.dart #1

```

45 floatingActionButton: FloatingActionButton(
+ 46   child: Icon(Icons.add),
47   onPressed: () {
48     setState(() => loading = true);
49     Navigator.push(context, MaterialPageRoute(builder: (context) => Edit(note: new Note()))).then((v) {
50       refresh();
51     });
52   }
53 ), // FloatingActionButton
54 body: loading? Loading() : ListView.builder(
55   padding: EdgeInsets.all(5.2),
56   itemCount: notes.length,
57   itemBuilder: (context, index) {
58     Note note = notes[index];
59     return Card(
60       color: Colors.white70,
61       child: ListTile(
62         title: Text(note.title),
63         subtitle: Text(
64           note.content,
65           maxLines: 2,
66           overflow: TextOverflow.ellipsis,
67         ), // Text
68         onTap: () {
69           setState(() => loading = true);
70           Navigator.push(context, MaterialPageRoute(builder: (context) => Edit(note: note))).then((v) {
71             refresh();
72           });
73         },
74       ), // ListTile
75     ); // Card
76   }, // ListView.builder
77 ); // Scaffold
78
79
80 }
81
82 Future<void> refresh() async {
83
84   notes = await DB().getNotes();
85   setState(() => loading = false);
86
87 }
88
89 }

```

Figure 18: Code - notes_log.dart #2

- lib > views > update_note.dart:

```
lib > views > update_note.dart > ...
3  import 'package:flutter/material.dart';
4  import 'package:todoist_app/designs/note_design.dart';
5  import 'package:todoist_app/process/database.dart';
6  import 'package:todoist_app/widgets/load.dart';
7
8  class Edit extends StatefulWidget {
9
10     final Note note;
11     Edit({ this.note });
12
13     @override
14     EditState createState() => EditState();
15
16 }
17
18 class EditState extends State<Edit> {
19
20     TextEditingController title, content;
21     bool loading = false, editview = false;
22
23     @override
24     void initState() {
25         super.initState();
26         title = TextEditingController();
27         content = TextEditingController();
28         if(widget.note.id != null) {
29             editview = true;
30             title.text = widget.note.title;
31             content.text = widget.note.content;
32         }
33     }
34
35     @override
36     Widget build(BuildContext context) {
37
38         return Scaffold(
39             appBar: AppBar(
40                 title: Text(editview? 'Edit Note' : 'New Note'),
41                 actions: <Widget>[
42                     IconButton(
43                         icon: Icon(Icons.save),
44                         onPressed: () {
45                             setState(() => loading = true);
46                             save();
47                             Navigator.pop(context);
48                         },
49                     ), // IconButton
```

Figure 19: Code - update_note.dart #1

```

50         if(editview) IconButton(
51             icon: Icon(Icons.delete),
52             onPressed: () {
53                 setState(() => loading = true);
54                 delete();
55             },
56         ), // IconButton
57     ], // <Widget>[]
58 ), // AppBar
59 body: loading? Loading() : ListView(
60     padding: EdgeInsets.all(15.0),
61     children: <Widget>[
62         TextField(
63             controller: title,
64             decoration: InputDecoration(
65                 hintText: 'Enter Title'
66             ), // InputDecoration
67         ), // TextField
68         SizedBox(height: 18.0),
69         TextField(
70             controller: content,
71             decoration: InputDecoration(
72                 hintText: 'Enter Content'
73             ), // InputDecoration
74             maxLines: 10,
75         ), // TextField
76     ], // <Widget>[]
77 ), // ListView
78 ); // Scaffold
79
80 }
81
82 Future<void> save() async {
83     if(title.text != '') {
84         widget.note.title = title.text;
85         widget.note.content = content.text;
86         if(editview) await DB().update(widget.note);
87         else await DB().add(widget.note);
88     }
89     setState(() => loading = false);
90 }
91
92 Future<void> delete() async {
93     await DB().delete(widget.note);
94     Navigator.pop(context);
95 }
96
97 }

```

Figure 20: Code - update_note.dart #2

- lib > views > search.dart:

```

lib > views > search.dart > ...
3  import 'package:flutter/material.dart';
4  import 'package:todoist_app/designs/note_design.dart';
5  import 'package:todoist_app/process/database.dart';
6  import 'package:todoist_app/widgets/load.dart';
7
8  class Search extends StatefulWidget {
9    @override
10   _SearchState createState() => _SearchState();
11 }
12
13 class _SearchState extends State<Search> {
14
15   TextEditingController _textEditingController = TextEditingController();
16
17   List<Note> notesOnSearch = [];
18   List<Note> notes;
19   bool loading = true;
20
21   @override
22   void initState() {
23     super.initState();
24     refresh();
25   }
26
27   @override
28   Widget build(BuildContext context) {
29
30     return Scaffold(
31       appBar: AppBar(
32         title: Container(
33           decoration: BoxDecoration(color: Colors.blue.shade100),
34           child: TextField(
35             onChanged: (value) {
36               setState(() {
37                 notesOnSearch =
38                   notes.where((element) => element.toString().toLowerCase().contains(value.toLowerCase())).toList();
39               });
40             },
41             controller: _textEditingController,
42             decoration: InputDecoration(
43               border: InputBorder.none,
44               errorBorder: InputBorder.none,
45               enabledBorder: InputBorder.none,
46               focusedBorder: InputBorder.none,
47               contentPadding: EdgeInsets.all(5),
48               hintText: "Search Notes",
49             ), // InputDecoration
50           ), // TextField
51         ), // Container
52       ), // AppBar
53       body: loading? Loading() : ListView.builder(
54         padding: EdgeInsets.all(5.2),
55         itemCount: _textEditingController.text.isNotEmpty ? notesOnSearch.length : notes.length,
56         itemBuilder: (context, index) {
57           Note note = _textEditingController.text.isNotEmpty ? notesOnSearch[index] : notes[index];
58           return Card(
59             color: Colors.white70,
60             child: ListTile(
61               title: Text(note.title),
62               subtitle: Text(
63                 note.content,
64                 maxlines: 2,
65                 overflow: TextOverflow.ellipsis,
66               ), // Text
67             ), // ListTile
68           ); // Card
69         }, // ListView.builder
70       ); // Scaffold
71     );
72   }
73
74   Future<void> refresh() async {
75
76     notes = await DB().getNotes();
77     setState(() => loading = false);
78   }
79
80 }
81
82 }

```

Figure 21: Code - search.dart

- lib > designs > note_design.dart:

```
lib > designs > note_design.dart > ...
5  class Note {
6
7      int id, date;
8      String title, content;
9
10     setDate() {
11         DateTime now = DateTime.now();
12         String ds = now.year.toString() + now.month.toString() + now.day.toString() + now.hour.toString()
13         + now.minute.toString() + now.second.toString();
14         date = int.parse(ds);
15     }
16
17     Note();
18
19     Note.fromMap(Map<String, dynamic> map) {
20         id = map['id'];
21         date = map['date'];
22         title = map['title'];
23         content = map['content'];
24     }
25
26     toMap() {
27
28         return <String, dynamic>{
29             'id': id,
30             'date': date,
31             'title': title,
32             'content': content,
33         };
34     }
35
36 }
```

Figure 22: Code - note_design.dart

- lib > process > database.dart:

```
lib > process > database.dart > ...
4  import 'package:sqflite/sqflite.dart';
5  import 'package:path/path.dart';
6  import 'package:todoist_app/designs/note_design.dart';
7
8  class DB {
9
10     static Database _database;
11     final String table = 'notes';
12
13     Future<Database> get db async {
14         if(_database != null) return _database;
15         _database = await initDB();
16         return _database;
17     }
18
19     initDB() async {
20         var dir = await getDatabasesPath();
21         String path = join(dir, 'noteup.db');
22         var database = await openDatabase(
23             path,
24             version: 1,
25             onCreate: (Database _db, int version) async {
26                 await _db.execute(
27                     'CREATE TABLE $table(id INTEGER PRIMARY KEY AUTOINCREMENT, date INTEGER, title TEXT, content TEXT)');
28             });
29     }
30
31     return database;
32 }
33
34 Future<void> add(Note note) async {
35     var database = await db;
36     note.setDate();
37     await database.insert(table, note.toMap());
38 }
39
40 Future<void> update(Note note) async {
41     var database = await db;
42     note.setDate();
43     await database.update(table, note.toMap(), where: 'id = ?', whereArgs: [note.id]);
44 }
45
46 Future<void> delete(Note note) async {
47     var database = await db;
48     await database.delete(table, where: 'id = ?', whereArgs: [note.id]);
49 }
50
51 Future<List<Note>> getNotes() async {
52     var database = await db;
53     List<Map<String, dynamic>> maps = await database.rawQuery('SELECT * FROM $table ORDER BY date DESC');
54     List<Note> notes = List<Note>.empty(growable: true);
55     for(Map<String, dynamic> map in maps) {
56         notes.add(Note.fromMap(map));
57     }
58     return notes;
59 }
60
61 }
```

Figure 23: Code - database.dart

- lib > widgets > load.dart:

```
lib > widgets > load.dart > ...
1  import 'package:flutter/material.dart';
2
3  class Loading extends StatelessWidget {
4
5      @override
6      Widget build(BuildContext context) {
7          return Center(
8              child: CircularProgressIndicator(
9                  strokeWidth: 3.0,
10             ), // CircularProgressIndicator
11         ); // Center
12     }
13
14 }
```

Figure 24: Code - load.dart

2.6 Test Case

Test Scope:

The major scope of the current testing design is to make sure that ToDo-List mobile application meets the proper requirements.

Test Objectives:

- Test essential parts of the application for proper functionalities.
- Check if the requirements correspond with the developed application.
- Check the efficiency and effectiveness of the application.

Test Approach:

The current testing design for the ToDo-List mobile application;

1. Uses a common test case template for each test case.
2. Test fundamental structures and functions of the application.
3. Test one section of every essential task in the database system.

Test Case Template:

Table 2: Test case template

Test Description				
Test Case	Input Data	Expected Outcome	Actual Outcome	Outcome Result (Pass/Fail)

Testing Targets:

Table 3: Testing targets

Test Explanation		
Test Case ID	Test Description	Test Date
01	View list of existing notes.	28/12/2021
02	Create new note.	28/12/2021
03	Edit existing note.	28/12/2021
04	Delete existing note.	28/12/2021

Test Implementation:

Table 4: Test Case 01 - View list of existing notes

Test Description		View list of existing notes.		
Test Case	Input Data	Expected Outcome	Actual Outcome	Outcome Result (Pass/Fail)
01	-	View all existing notes.	View all existing notes.	PASS

Evidence:

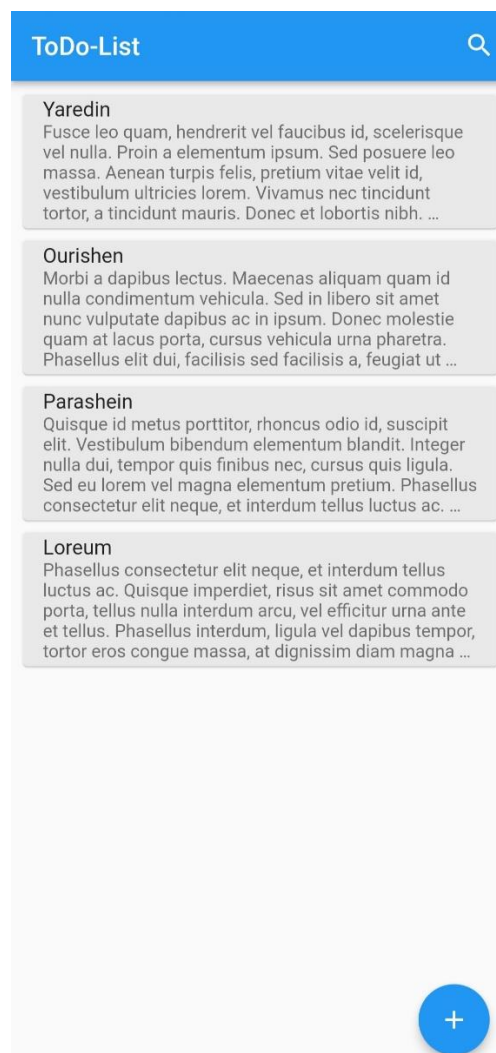


Figure 25: View list of existing notes - Test Evidence

Table 5: Test Case 02 - Create new note

Test Description		Create new note.		
Test Case	Input Data	Expected Outcome	Actual Outcome	Outcome Result (Pass/Fail)
02	Title: "Test title" Content: "Sample content for test"	Note added to database.	Note added to database.	PASS

Evidence:

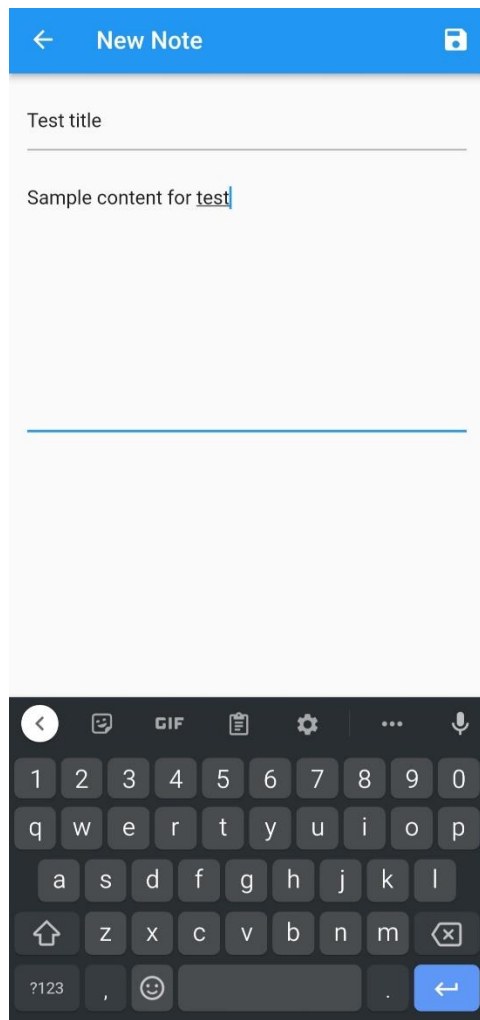


Figure 27: Create new note - Test Evidence #1

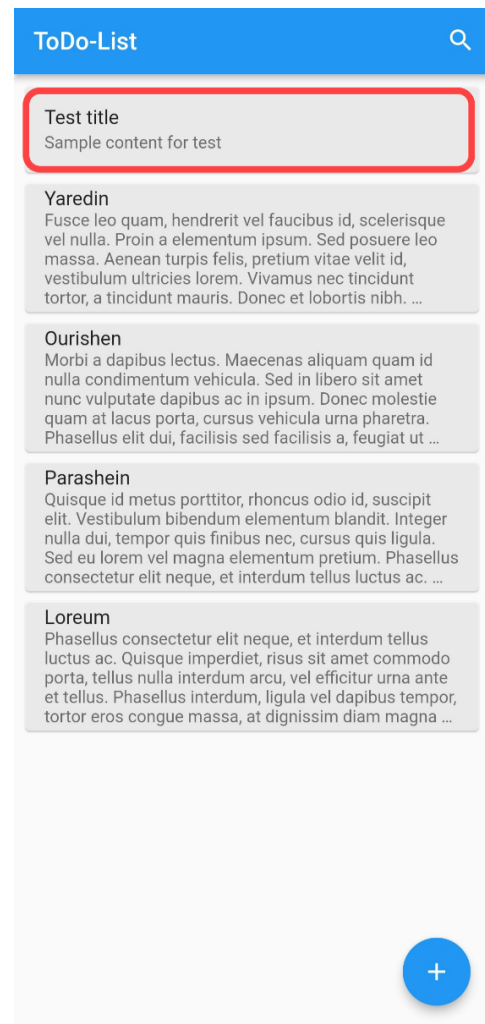


Figure 26: Create new note - Test Evidence #2

Table 6: Test Case 03 - Edit existing note

Test Description		Edit existing note.		
Test Case	Input Data	Expected Outcome	Actual Outcome	Outcome Result (Pass/Fail)
03	Title: "Test title EDITED now" Content: "Sample content for test EDITED now"	Note values updated successfully.	Note values updated successfully.	PASS

Evidence:

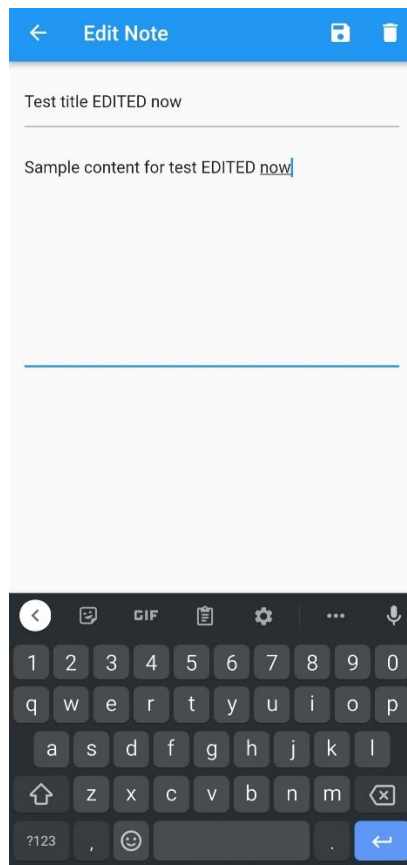


Figure 29: Edit existing note - Test Evidence #1

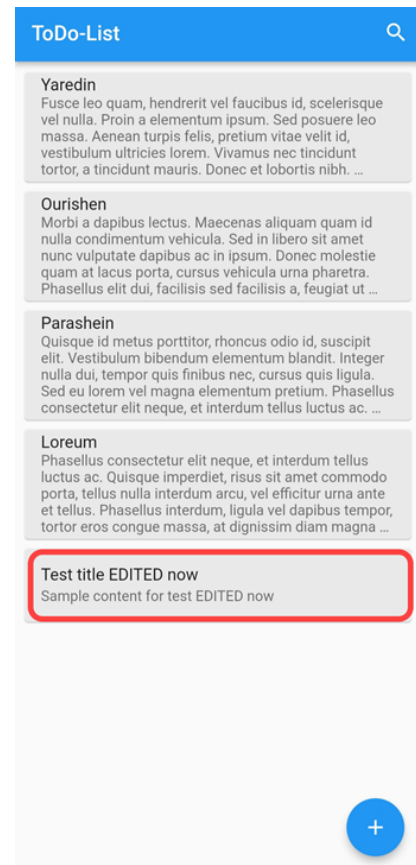


Figure 28: Edit existing note - Test Evidence #2

Table 7: Test Case 04 - Delete existing note

Test Description		Delete existing note.		
Test Case	Input Data	Expected Outcome	Actual Outcome	Outcome Result (Pass/Fail)
04	-	Note deleted from database successfully.	Note deleted from database successfully.	PASS

Evidence:

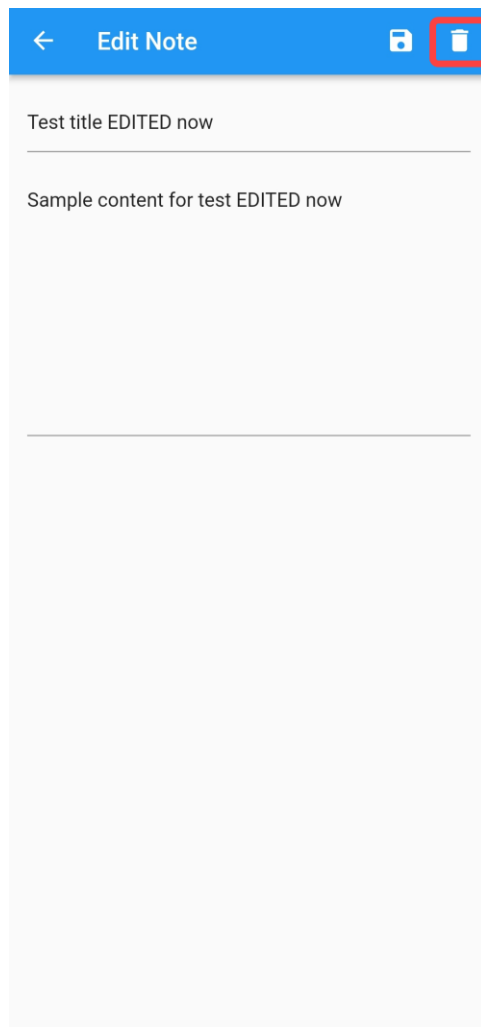


Figure 31: Delete existing note - Evidence #1



Figure 30: Delete existing note - Evidence #2

References

- [1] I. Wigmore, "What is mobile app?," December 2013. [Online]. Available: <https://whatis.techtarget.com/definition/mobile-app>. [Accessed 23 November 2021].
- [2] A. S. Gillis, "What is native app?," August 2020. [Online]. Available: <https://searchsoftwarequality.techtarget.com/definition/native-application-native-app>. [Accessed 24 November 2021].
- [3] "What is Cross-Platform App Development?," 23 June 2020. [Online]. Available: <https://fullscale.io/blog/what-is-cross-platform-app-development/#:~:text=Cross%2Dplatform%20mobile%20applications%20are,such%20as%20iOS%20and%20Android..> [Accessed 24 November 2021].
- [4] mobiThinking, "What is a Web-based mobile application or Web app?," 12 August 2010. [Online]. Available: <https://mobiforge.com/news-comment/what-a-web-based-mobile-application-or-web-app-heres-expert-opinion-w3c>. [Accessed 24 November 2021].
- [5] T. Contributor, "What is hybrid application (hybrid app)?," September 2019. [Online]. Available: <https://searchsoftwarequality.techtarget.com/definition/hybrid-application-hybrid-app>. [Accessed 24 November 2021].
- [6] N. Sakovich, "Cross-Platform Mobile Development:," May 2018. [Online]. Available: <https://www.sam-solutions.com/blog/cross-platform-mobile-development/>. [Accessed 24 November 2021].
- [7] javaTpoint, "What is Dart Programming," [Online]. Available: <https://www.javatpoint.com/flutter-dart-programming>. [Accessed 28 December 2021].
- [8] G. Thomas, "What is Flutter and Why You Should Learn it in 2020," freeCodeCamp, 12 December 2019. [Online]. Available: <https://www.freecodecamp.org/news/what-is-flutter-and-why-you-should-learn-it-in-2020/>. [Accessed 28 December 2021].
- [9] N. Sakovich, "Cross-Platform Mobile Development: Five Best Frameworks," 22 June 2018. [Online]. Available: <https://www.sam-solutions.com/blog/cross-platform-mobile-development/>. [Accessed 28 December 2021].
- [10] SQLiteTutorial, "What Is SQLite," [Online]. Available: <https://www.sqlitetutorial.net/what-is-sqlite/>. [Accessed 28 December 2021].